Remarks

Status of application

Claims 1-47 were examined and stand rejected in view of prior art. The claims have been amended to further clarify Applicant's invention. Reexamination and reconsideration are respectfully requested.

The invention

A system and methodology for cross language type system compatibility is described. In one embodiment, for example, a system of the present invention for translation of data types between a first application in a first language and a second application in a second language is described that comprises: a computer having at least one processor and a memory; a formal mapping between data types of the first language and data types of the second language; translators for translating data types between the first language and the second language based on the formal mapping; a translation mapping to the translators based on actual data types of the first application and formal data types of the second application; and a module for automatically selecting an appropriate translator for translating between a particular data type in the first language and a data type in the second language based on the translation mapping in response to invocation of a method of the first application with the particular data type.

In another embodiment, for example, a method of the present invention is described for translation of data types between a first component in a first language and a second component in a second language, the method comprises steps of: defining a formal mapping between data types of the first language and data types of the second language; implementing translators based on the formal mapping for translating data types between the first language and the second language; producing a programming interface for the first component based upon the formal mapping and the second component's programming interface; generating a translation mapping to the translators based on actual data types of the first component and formal data types of the second component as defined in the first component's programming interface; in response to invocation of a method defined in the first component's programming interface with a particular data type, automatically selecting a translator based on the translation mapping and the particular data type; and translating the particular data type to a data type of the second language using the selected translator.

General

A. Specification objections

The Examiner objects to the Specification at paragraph [0047] as containing a hyperlink reference. The paragraph has been amended to remove the reference.

The Examiner also objects to the Specification at paragraph at [0057] for use of the trademark Java^{TM}. The term has been tagged to indicate the provider's (Sun Microsystems) trademark claim. (Applicant takes no position as to whether third-party product names function as trademarks.)

However, the Examiner's insistence that Java be spelled JAVA is not understood, as that is not the way that Sun Microsystems employs the mark. Refer to the following Sun Microsystems' material provided in Applicant's Information Disclosure Statement filed September 18, 2006:

GOSLING, J. ET AL., Java Language Specification, 2nd Ed.(Chapters 1-5), Sun Microsystems, Inc., Mountain View, CA, June 2000.

GOSLING, J. ET AL., The Java Language Environment: A White Paper, Sun Microsystems Computer Company, October 1995

In both of these Sun Microsystems documents, Java is spelled "Java"; it is not spelled "JAVA". Other Sun Microsystems documents (e.g., JDBC Specification) also show that Sun Microsystems spells Java as "Java". In view of this, the Examiner's insistence that Java be spelled in all caps as "JAVA" is not understood.

B. Claims objections

Claims 46 and 47 stand objected to under 37 CFR 1.75(c), as being of improper dependent form for failing to further limit the subject matter of a previous claim. Both claims have been amended to further limit the subject matter of a previous claim.

C. Section 101 objection

Claims 1-21 and 47 stand rejected under 35 U.S.C. 101 on the basis that the claimed invention is directed to non-statutory subject matter. Independent claim 1 has been amended to recite that the claimed system includes "a computer having at least one processor and a memory." Claim 47 has been amended as previously described, and that amendment specifies that the claimed method steps are operating in a computer (i.e., a device). As such, it is respectfully submitted that the amended claims do not recite "pure software."

D. Claims 2, 44 and 45

Claims 2, 44 and 45 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite, for use of the term "Java". The claims had been amended to eliminate usage of "Java", thus overcoming the objection.

Prior art rejections

A. Section 102 rejection: Kanamori

Claims 1, 3-5, 7-22, 24-27, 29-43, 46 and 47 stand rejected under 35 U.S.C. 102(b) as being anticipated by Kanamori (US Patent No.6, 167,565). The Examiner's rejection of claim 1 is representative:

> As per claim 1, Kanamori discloses A system for translation of data types between a first application in a first language and a second application in a second language, the system comprising (c3:64-c4:49):
>
> a formal mapping between data types of the first language and data types of the second language (for example, c4: 3-6, " ... a mapping from a data type in one programming language to a corresponding data type in another programming language...");
>
> translators for translating data types between the first language and the second language based on the formal mapping; a translation mapping to the translators based on actual data types of the first application and formal data types of the second application (c4: 6-9, "This mapping

specifies custom marshaling code...that can be used for converting...data type to ...the corresponding data type... ""); and

a module for selecting an appropriate translator for translating between a particular data type in the first language and a data type in the second language based on the translation mapping in response to invocation of a method of the first application with the particular data type (c3: 65-c4: 2, "...customer marshaling of parameters during invocations...").

For the reasons stated below, Applicant's claimed invention may be distinguished on a variety of grounds.

Kanamori describes an approach providing custom marshaling of parameters during invocation of functions implemented in a second language by computer programs implemented in a first language. The parameter to be custom marshaled has a first type in the first language and a second type in the second language. As Kanamori points out (Kanamori Col. 3, Lines 5-10), prior art programming language translators have been developed to map the data types of one programming language to the data types of another programming language when a function implemented in one programming language is invoked by a computer program implemented in another programming language. The particular problem addressed by Kanamori is the following (Kanamori Col. 3, Lines 18-22): these prior art translators only converted between types for which it happens to have predefined marshaling code. If a programmer defines a complex data type, then the translator is unlikely to be able to perform the needed conversions automatically. To address this problem, Kanamori provides a custom marshaling system that receives custom code (i.e., provided by the programmer user) that may be executed for converting a passed parameter of one type to a parameter of another type (see, e.g., Kanamori at Col. 3, Lines 25-45). Kanamori's approach allows a programmer to extend data type conversation otherwise provided by the (built-in) translator but requires the programmer to do the "heavy lifting" of providing custom marshaling code (i.e., a set of instructions authored by the programmer) to be used for converting parameters of that data type to a parameter of the corresponding data type, and vice versa. Such an approach places too heavy a burden on the programmer, who must now be concerned about providing custom marshaling code, in addition to the chore of developing the underlying application program (which is to use the custom marshaling code). This is not the approach invented and claimed by Applicant.

Both Kanamori's approach and Applicant's approach address the challenge in developing programs in multiple language environments -- that each language may include specialized data types. Data types defined in one language may be incompatible with those in another language. As described in Applicant's Specification (Paragraph [0058]), prior art solutions for mapping data from data types of one language to another language have traditionally relied on a one-to-one mapping between data types. In this regard, Kanamori appears to be little changed from other prior art systems. This isomorphic mapping requirement limits the flexibility and usability of such systems, in that the developer developing a program in one language must be aware of the type system requirements of the other language (or languages) being used.

Note that in Kanamori's system, during execution of the computer program when the inter-language invocation is encountered, the translator retrieves the specified custom marshaling code for the parameter and executes that code to convert the actual parameter into the data type of the formal parameter. Here, Kanamori's approach requires the programmer to come up with the custom marshaling code. The user of Kanamori's system (i.e., the programmer) must be aware of the type system requirements of the other language(s) being used, so that he or she can write and/or otherwise provide the custom code that is to be received by the Kanamori custom marshaling system.

Applicant's invention enables a programmer developing a program in this type of multiple language environment to use the familiar, native data types of a given programming language without concern about the type system requirements of other languages. Applicant's invention provides for translation of data types from a "source" language to a "target" language (and vice versa) in a manner that does not place restrictions on the data types that can be used in the individual languages and does not force a developer working in one language to use the data types of another language. More particularly, Applicant's invention provides an automated mechanism for determining the optimal data type to provide, given (a) the formal data type (e.g., the data type required by the target) and (b) the actual data type (e.g., the data type provided by the source). The operations of the Applicant's invention are transparent to the user (e.g., a developer writing an application in a given programming language). The developer may write a component using the native data types of a given language, without any need to be concerned about the corresponding data types in other languages. For example, a component written in C# may pass a list of customers to another application component written in Java, notwithstanding the fact that the C# and Java components may have different data types for representing this list of customers. The system of the Applicant's invention provides an intermediary -- a data translation layer -- for automatically translating the data types from a source language (e.g., C#) to a target language (e.g., Java) in a manner that is flexible and correct. Significantly, the system automatically identifies (not the user programmer) the optimal target data type in the target language (e.g., Java) that should be used for translation based upon the data type provided by the source language (e.g., C#). This is very different from Kanamori's manual approach, which requires the programmer to provide custom code (i.e., written by the programmer himself or herself) to carry out any such conversion operation.

Turning now to more specific features of Applicant's invention, as described in Applicant's Specification (e.g., at Paragraphs [0061-0066]), Applicant's invention employs a data translation layer that determines the appropriate target (e.g., Java) data types (given a source object of the other type) and then appropriately translates data to the appropriate target data type. In order to appreciate its operation and functionality, consider the following example usage of Applicant's invention.

Suppose a client application written in the C# language wishes to invoke a method or service provided by a server component written in the Java language. As such, the server component uses a different type system than the client application. Further suppose that the client application wishes to invoke the service of the server and pass data defined as a C# data type (e.g., a C# list). The server, on the other hand, is expecting a Java list. The data translation layer of Applicant's invention serves as an intermediary between the two to automatically carry out this task. Here, the data translation layer determines how best to translate from one to another and performs the appropriate translation. For example, when invoked by the client with a C# list, the data translation layer initially determines the appropriate target data type (i.e., the Java data type). The data translation layer reads the client data type (e.g., C# list) into an internal format, and then writes the data to the appropriate server data type (e.g., Java list) from the internal format. Of particular interest, during this process the data translation layer determines the optimal reader/writer object to use for the translation which provides the best mapping between the data types of the two languages.

Note that, in contrast to Kanamori's approach, the programmer using Applicant's system need not write any custom marshalling code to do this conversion. Instead this is done automatically in Applicant's system. Whereas Kanamori's approach is one where the programmer is expected to do the "heavy lifting" (of writing custom conversion code), in Applicant's approach the heavy lifting is done by core converter objects present in the data translation layer that are automatically selected based on data type. Each core converter object serves as a "Translator" object that automatically construct a data value in one type system given a data value in another type system (and vice versa). When sending a data value from the client language to the server language, the methodology of the Applicant's invention initially provides for looking for a Translator object corresponding to the combination of the actual type (source type) and the formal type (target type). If a match is found, that Translator object is used for the data conversion. However, if a match is not found, a search is performed through all of the supported types of the actual type (that is, in Java or C# search through all interfaces supported by the actual type, and all base classes of the actual type). For each supported type of the actual type (source type), Applicant's system looks for a Translator object corresponding to the combination of the supported type of the actual type and the formal type (target type). This search proceeds until a match is found (at which point that Translator is used for the data conversion), or until it is determined that no possible match exists (at which point a failure occurs, indicating that the required data conversion is not possible).

The distinguishing features of Applicant's invention highlighted above are set forth in Applicant's independent claims. For example, claim 1 includes the claim limitation of:

> a module for automatically selecting an appropriate translator for translating between a particular data type in the first language and a data

> type in the second language based on the translation mapping in response to invocation of a method of the first application with the particular data type.

(Independent claim 22 includes a similar claim limitation.) By providing an automated means to do this conversion, Applicant's approach provides a significant improvement over the prior art. With Applicant's approach, if a programmer defines a complex data type in one language, the appropriate translator is automatically selected to perform the needed conversion(s) into another language, and do so in a manner that is transparent to the programmer. Such a feature is not taught or suggested by Kanamori, which clearly requires the programmer to write custom code to perform such a conversion. Moreover, as a given data type of a source language may possibly map to a plurality of data types of a target language, Applicant's invention provides for selecting the <u>optimal</u> target data type given a particular source data type.

For the reasons stated, it is respectfully submitted that Applicant's invention is not taught or suggested by Kanamori. If anything, Kanamori's approach of requiring the programmer to write <u>custom</u> conversion code teaches away from Applicant's invention. Moreover, the unique feature of automatically selecting the appropriate translator upon method invocation is set forth as an affirmative claim limitation in all of Applicant's independent claims, thus rendering them and their dependents distinguishable over the art. Accordingly, it is respectfully submitted that any rejection under Section 102 is overcome.

B. Section 103 rejection: Kanamori and Vargas

Claims 2, 44 and 45 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Kanamori (US Patent No.6, 167,565) in view of Vargas (US PGPub. No. 2004/0103405). Here, the Examiner largely repeats the above rejection based on Kanamori but adds Vargas for the notion that it teaches conversion of C# as one of the programming languages. The claims are believed to be allowable for at least the reasons cited above pertaining to the Section 102 rejection based on Kanamori. In particular, Applicant's claimed invention provides an approach where an appropriate translator is automatically selected upon method invocation. Kanamori does not teach this feature and instead teaches (away) that the programmer should manually provide this conversion code. Nothing in Vargas remedies or overcomes this core deficiency of Kanamori.

C. Section 103 rejection: Kanamori and Beisiegel

Claims 6, 23 and 28 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Kanamori (US Patent No.6, 167,565) in view of Beisiegel et al. (US PGPub. No. 2004/0177360, hereinafter "Beisiegel"). Here, the Examiner largely repeats the above rejection based on Kanamori but adds Beisiegel for the notion that it teaches Applicant's claim limitation pertaining to the translators marshalling translated data into a wire format for transfer across a network. The claims are believed to be allowable for at least the reasons cited above pertaining to the Section 102 rejection based on Kanamori. As discussed above, Applicant's claimed invention provides an approach where an appropriate translator is automatically selected upon method invocation. Kanamori does not teach this feature and instead teaches away -- that the programmer should manually provide this conversion code. Nothing in Beisiegel remedies or overcomes this core deficiency of Kanamori.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 408 884 1507.

Respectfully submitted,

Date: July 17, 2008

/John A. Smart/

John A. Smart; Reg. No. 34,929
Attorney of Record

408 884 1507
815 572 8299 FAX